Veyrack LIN
Maxence BRUNET

Enseignants : *Frédéric* Peschanski
Romain Demangeon

UE PAF
Compte-Rendu

Juin 2020

Science et Technologie du Logiciel
Sorbonne Université
Année 2019/2020

Contents

Introduction

L'objectif de ce projet est le développement d'un jeu vidéo sûr en Haskell. Le jeu est en 2d et a pour thème l'univers du Donjon Crawling. A l'aide de différentes ressources, que ce soit la bibliothèque SDL2 ou encore des tileset libre de droits, nous avons mis en place un Donjon Crawler utilisant une boucle non-réactive. Dans la suite de ce rapport, nous allons prendre le temps d'expliquer le mode d'emploi du jeu, la description de la structure complète du jeu et les extensions qui ont été réalisées. On expliquera de manière succinctes les prépositions et les test mis en place. Enfin nous parlerons des choix d'implémentations qui peuvent sembler surprenants mais qui ont en réalité du sens.

1 Manuel utilisation

1.1 Les instructions de lancement

Les instructions pour lancer le jeu :
stack run
Les instructions pour lancer les test :
stack test

1.2 Les instructions "in-game"

Le déplacement du personnage:

- Les touches *"Z", "Q", "S", "D"*

L'interaction avec les entités du donjon:

- La touche *"E"* pour ouvrir des portes ou des coffres.
- Les portes ouvertes sont de couleurs bleu et celles fermées de couleurs rouge.
- La vie est affichée au dessus du personnage (en noir).
- Le personnage perd de la vie s'il tombe dans un piège.
- Si le personnage perd toute sa vie, la partie est perdue.
- Si le personnage se déplace sur la sortie avec le trésor, la partie est gagnée.
- La touche *"A"* pour utiliser une potion qui rend de la vie. Au moins une potion doit être présente dans l'inventaire du personnage.

2 La carte du jeu

L'environnement où le personnage évolue est situé dans le fichier **carte.hs**. Toutes les fonctions qui touchent à la carte sont situés dans ce fichier.

2.1 Structure utilisé pour la carte

```haskell
data Statut = Ouvert | Ferme deriving (Eq,Show)

data Direction = NS | EO deriving (Eq,Show)

data Case = Vide
  | Porte Direction Statut
  | Mur
  | Coffre Statut
  | Tresor Statut
  | Entree
  | Sortie
  | Zombie
  | Pique Statut
  | ClotureElectrique Direction Statut
  | Levier Statut
  deriving (Eq, Show)

data Coord = Coord { cx :: CInt, cy :: CInt} deriving (Eq,Show,Ord)

data Terrain = Terrain {ht :: CInt, lg :: CInt, contenu :: (M.Map Coord Case)}
    deriving (Show,Eq) --ht = hauteur , lg = largeur
```

La structure utilisée est similaire à celle proposé dans le sujet. Le terrain est l'environnement dans lequel le personnage évolue. Les valeurs *ht* et *lg* représentent la hauteur et la largeur du donjon. Sa dernière valeur "contenu" représente la carte. Toutes les entités présentes dans l'environnement y sont présentes, à l'exception de notre personnage. La carte (alias contenu) est une map qui prend comme clé une coordonnée et comme valeur une case. La coordonnée est représenté par deux points (Jeu en 2 dimensions). Une case est représente par une des entités du donjon. Certaines entités peuvent effectuer des actions, d'où la création d'un record "Statut". Elles peuvent également être positionné dans une direction précise, d'où la création d'un record "Direction".

Il existe la possibilité de définir son propre donjon. Pour cela, il suffit de se rendre dans le dossier "CarteGenerator" et de modifier le fichier **carte.txt**.

2.2 Ajouts Possibles

Il est possible d'ajouter des entités représentées par les caractères suivant:

- Créer un Mur : caractère 'x'
- Créer un Coffre : caractère 'c'
- Créer une Porte (Fermée) : caractère '-' pour des portes NS. Caractère '|' pour des portes EO
- Créer une Entrée : caractère 'E'. Cette entrée est le point de départ du personnage (non visible).
- Créer un Zombie : caractère 'z'. Cette entité ennemi n'attaque pas.
- Créer une Sortie: caractère 'S'.
- Créer un Pique (Piège) : caractère 'p'.
- Créer une clôture électrique (En marche) : caractère 'w' pour les clôtures NS et 'k' pour le clôtures EO.
- Créer un Levier : caractère 'l'. Permet de désactiver la clôture électrique la plus proche du levier actionné.
- Créer un Trésor : caractère 't'.

2.3 Conditions sur les ajouts

Malgré la liberté laissée pour la construction de la map, il y a certaines règles à respecter :

- Les murs du donjon doivent former un rectangle.
- Toutes les entités (Coffres, portes etc..) doivent être à l'intérieur du donjon.
- Les portes doivent être situés entre 2 murs.
- Il existe un seul et unique trésor qui permet de sortir du donjon.
- Il doit y avoir autant de leviers que de clôtures électriques

Si l'une de ces conditions n'est pas respectée, on déclenche une exception crée par nos soins. L'exception crée se situe dans le fichier **Exception.hs** :

```
data CustomException = InvalidMapException
                     | Autreexception
                     deriving(Show)
instance Exception CustomException
```

2.4 Les corrections de bugs

Nous pensons qu'il est important d'intégrer dans le jeu un système de debug visible pour permettre à un utilisateur de modifier ou d'ajouter des éléments au jeu.
Dans la boucle du jeu (gameLoop dans **main.hs**), un champ "Display debug" a été ajouté pour voir apparaître à l'écran un carré rouge. Ce carré représente la zone de collision du personnage. Il est également possible de commenter le champ "Display Brouillard de guerre" pour voir l'intégralité de la map (utile pour le level design).

```
--Display Brouillard de guerre
  displayAura renderer tmap smap tx ty
  displayFog renderer tmap smap 0 0 (ht*tailleBloc) (lg*tailleBloc)
                                        (fromIntegral (tx)) (fromIntegral (ty))

-- |Display debug
 displayDebug renderer
```

3 Le modèle du jeu

La majeure partie du code source se situe dans le ficher **Model.hs**. Il contient l'ensemble des actions du personnage et ses interactions avec l'environnement qui l'entoure.

3.1 Structure utilisé pour le modèle

Avant d'aller plus loin, il est important d'expliquer la construction de notre modèle. Cette fois-ci, la structure utilisées dérive de celle présentée dans le sujet de départ:

```
data Translation = Translation { transX :: CInt
                               ,transY :: CInt
                               }
                               deriving (Show,Eq)

data Perso = Perso { persoX :: CInt
                   ,persoY :: CInt
                   ,direction :: DirectionPerso
                   ,vie :: CInt
                   ,inventaire :: Map Item CInt
                   }
                   deriving (Show,Eq)

data Item = Potion
          | Masque
          deriving (Show,Eq,Ord)
```

```haskell
data EtatJeu = Gagner
             | Perdu
             | Encours
             deriving (Show,Eq)

data GameState = GameState { translate :: Translation
                           ,tour :: Int
                           ,speed ::CInt
                           ,perso :: Perso
                           ,terrain :: Terrain
                           ,etatjeu :: EtatJeu
                          }
                          deriving (Show, Eq)

data DirectionPerso = North | West | South | East deriving (Eq,Show)
```

Détaillons le GameState pour y voir plus clair. Dans un premier temps, nous possédons un élément "translate" représentant une translation de notre environnement. Ici, l'objectif est d'avoir un personnage centré sur l'écran mais qui, lorsqu'il se déplace, déplace l'environnement. Nous reviendrons plus loin sur les méthodes utilisées pour réaliser cette translation. Par la suite, nous avons "tour". Il représente le nombre de tour ayant eu lieu depuis le début du jeu jusqu'à la fin. Ensuite, nous avons "speed" qui représente la vitesse du personnage. Puis "perso" qui représente notre personnage jouable dans le jeu. Il possède 2 coordonnées (x et y), une direction (Nord, Sud, Est, Ouest), de la vie et un inventaire qui lui permet de récupérer des objets lors de son parcours du donjon. Le champ "terrain" a déjà été expliqué précédemment (1.3.3). Enfin, nous avons l'état du jeu "etatjeu" qui permet de connaître à tout moment l'état de la partie. Maintenant que la structure du modèle a été détaillé, nous allons pouvoir détailler le rôle de chacune des entités du jeu.

3.2 Description des entités du jeu

Notre jeu contient un ensemble d'entités qui semble important à expliquer pour comprendre le fonctionnement du jeu. La description est expliquée dans cette partie (Model du jeu) car toutes les fonctions d'interaction avec les entités se situent dans le modèle.

Tout d'abord, nous avons des murs. Les murs permettent la construction de notre donjon en délimitant la zone dans laquelle le joueur est autorisé à se déplacer. Nous avons une sortie (Respectivement une entrée) qui permet de terminer le jeu en cours ou d'accéder à une autre salle du donjon (dans le cas où il existe un deuxième niveau). Rappelons qu'il est obligatoire de trouver le trésor du donjon pour pouvoir en sortir. Ensuite, nous avons les portes qui sont deux types : les portes Nord-Sud et Est-Ouest qui peuvent être soit ouverte, soit fermée.
En ce qui concerne l'aspect jeu vidéo ludique, nous en avons mis en place un certain nombre:

- **Les coffres :** Permettent à un joueur de récupérer une potion redonnant de la vie.
- **Les zombies :** Qui se déplacent dans le donjon (L'implémentation du combat n'a pas été mis en place)
- **Les piques :** Piège qui provoque chez le joueur une perte de vie et une légère secousse visuelle (à défaut d'être sonore)
- **Les clôtures électriques :** Piège qui provoque les mêmes dommages (en terme de vie et secousse visuelle) que les piques. La différence avec un autre piège est que celui-ci est lié à une autre entité, le levier.
- **Les leviers :** Chaque leviers permet d'ouvrir la clôture électrique la plus proche (Un levier par clôture). Une fois le levier activé, le personnage ne peut plus interagir avec celui_ci.
- **Un trésor :** Un objet qui permet de sortir du donjon. Le sprite est similaire à celui d'un coffre pour permettre au joueur de mieux explorer le donjon.

Notre personnage peut interagir avec chacune de ces entités. Pour cela différents types de collision ont été mis en place.

3.3 Collisions

La collision (hitbox) de notre personnage est différente selon les entités rencontrées. La collision pour un mur résultera d'une hitbox plus grande pour notre personnage que celle de la sortie par exemple.
Pour certaines entités, la collision est stricte. Dès que le personnage touche le bord de l'une d'elles, il y a collision. Elle concerne les entités telles que les murs, les portes, les coffres, les zombies et les leviers.
Pour d'autres entités, la collision est plus souple. La détection a lieu lorsque le personnage s'approche du centre. C'est le cas pour la sortie, les piques et les clôtures électriques.

4 Les extensions

Pour ce projet, nous avons mis en place différentes extensions lié à l'univers du donjon Crawler.

Dans un premier temps, nous considérons que le déplacement par **translation** est l'une des extensions proposées:

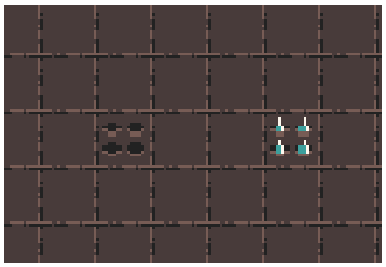
-Le personnage ne se déplace pas de case en case mais de pixel en pixel. En effet, le choix d'un déplacement par pixel rend le jeu plus fluide et plus agréable à prendre en main. Le personnage par défaut se déplace tous les 4 pixels mais peut-être modifier à l'initialisation du jeu (avec le champ speed). Ce déplacement par pixel nous a causé beaucoup de difficultés notamment pour les collisions qui sont effectuées au pixel près. Nous en reparlerons en détail dans la section "Choix d'implémentation".

Ensuite, nous avons mis en place un **système de vie** du personnage:

-Lorsque le personnage rencontre un piège, la vie située au dessus de celui-ci, diminue.

A propos des **pièges**, nous en avons mis quelques uns, tous différents les uns des autres :

-Les **piques** sont des pièges qui se déclenchent lorsque le personnage approche du centre. Les piques passent alors de l'état fermé à ouvert (voir image).



-Les **clôtures électriques** sont des pièges liés à un levier. Une clôture est active par défaut (statut = ouvert) et inflige des dommages au personnage s'approchant trop près du centre. Pour désactiver la clôture électrique, il faut actionner un levier. La clôture passe alors de l'état ouvert à fermé (voir image).

-Les **leviers** permettent de désactiver la clôture électrique la plus proche. Cela permet à l'utilisateur qui le souhaite de créer des niveaux avec plus de difficultés. Nous reviendrons

en détail sur l'approche utilisée pour les leviers dans la section "Choix d'implémentation".

Notre personnage peut perdre de la vie mais il peut aussi en récupérer :

-Les **Coffres** permettent de récupérer des potions utilisables à n'importe quel endroit du jeu. Pour cela nous avons mis en place un inventaire qui permet de déterminer le nombre de potions que le personnage possède. Les potions seront affichés en haut à gauche de l'écran.

Notre personnage doit posséder le trésor pour sortir du donjon:

-Le **trésor** du donjon est un masque. Pour celà, le personnage doit parcourir le donjon et ouvrir le coffre contenant le trésor. Lorsque le personnage possède le masque, il s'affiche en haut à gauche de l'écran.

Notre personnage se déplace sans voir l'intégralité de la carte:

-Le **Brouillard de guerre** à été mis en place pour apporter du challenge et être fidèle à l'univers du donjon crawler. Le personnage se déplace avec pour seul visibilité ce qui l'entoure.

5 Test et prépositions

5.1 Les tests

En ce qui concerne les test HSpec, il en existe de deux types. Ceux qui concernent la carte du jeu:

- getCaseFromString_spec: Vérifie que le string donné en entrée est soit une entité existante soit vide.
- updateKeyMap_spec: Vérifie qu'une clé appartenant à un objet dans une map a bien été mise à jour.
- updateValueMap_spec: Vérifie que la valeur d'un objet dans une map a été mise à jour sinon rajoute ce dernier dans la map.
- getCoordonneesObjectMap_spec: Vérifie qu'on récupère bien toutes les coordonnées d'une entité dans une map.
- collision_spec : Vérifie qu'une collision à bien lieu (on non).
- getSortie_spec : Vérifie qu'il existe une sortie.
- getEntree_spec : Vérifie qu'il existe une entrée.
- checkPorte_spec : Vérifie que les portes sont bien placées (Entre 2 murs).
- invariantMurs_spec : Vérifie que les murs du donjon forment un rectangle fermé.
- invariantObjets_spec : Vérifie qu'aucun objet est en dehors du donjon.

Et en ce qui concerne le modèle du jeu:

- changePv_spec: Vérifie que les points de vie du personnage ont bien été mise à jour.
- move_spec: Vérifie que le personnage s'est bien déplacé (vers le haut, bas, gauche ou

droite).
- openEntity_spec: Vérifie qu'une entité change de statut (Passe de l'état fermé à ouvert).
- openChest_spec: Vérifie que lors de l'ouverture d'un coffre, le personnage récupère une potion dans l'inventaire.
- testSortie_spec: Vérifie que le personnage peut prendre la sortie seulement s'il possède le trésor.
- testPiege_spec: Vérifie qu'un piège retire de la vie au personnage.

5.2 Quickcheck

Nous n'avons malheureusement pas trouvé de test QuickCheck à mettre en place. Nous n'avons pas trouvé d'éléments qui nous permet de générer une multitude de paramètres aléatoires vérifiant une propriété.

6 Les choix d'implémentations

Dans cette section, nous allons expliquer les différents choix d'implémentation.

- La translation :

L'utilisation de déplacement par pixel nous oblige parfois à mettre des valeurs en dur dans certaines fonctions. Cela ne pose aucun problème pour notre projet car les valeurs telles que la taille de la fenêtre, des cases et de nos entités sont définies par nous. Le problème peut être réglé en permettant à l'utilisateur une plus grande liberté de modifications (modifier la taille des cases et des entités en fonction des goûts de l'utilisateur) ce qui implique beaucoup de changement et autant de temps que la création du jeu en lui-même. Cela reviendrait à faire un "RPG maker". C'est d'ailleurs ce que l'on avait commencé à faire (voir début fichier **main.hs**) mais par manque de temps, l'idée a été suspendu.
Note: On permet néanmoins de changer la taille de fenêtre:

- Les collisions :

Pourquoi vérifier tous les bords du personnage et pas seulement les cases adjacentes?
Tout simplement pour une meilleure précision. Notre personnage se déplaçant tous les 4 pixels, il est possible qu'il se retrouve entre deux cases. Pour éviter de corriger des bugs par la suite, on a décidé d'opter pour cette méthode.
Pourquoi il existe plusieurs fonctions de collisions tel que "isitanEntity" et "isitanEntityFlex"?
Comme nous l'avons expliqué brièvement dans les parties précédentes, certaines entités possèdent des hitbox différentes les unes des autres. Nous avons catégorisé les entités dont les collisions sont strictes et celles dont la collision est au centre. Cela permet de reprendre

ce projet pour y ajouter plus facilement d'autres entités (ex: Ennemis agressif, projectiles, etc..).

- La carte :

Il n'existe pas de case vide dans la carte pourtant une case vide existe dans le records de la carte ?
Les cases vide sont des cases où il n'y a aucune interaction (Comme le sol). On a décidé de ne pas le rajouter à la carte pour éviter des calculs plus longs. Le parcours de la map est plus optimisé sans les cases vides. Néanmoins, il existe certaines conditions où une valeur de la map peut être vide. Si c'est le cas, on traite quand même son état. De plus, si on souhaite connaître la hauteur et la largeur de la carte, elle accessible à tout moment dans le record du Terrain.

- Levier et Clôture Électrique :

L'utilisation de ces deux entités a été pensé pour rendre les niveaux du donjon un peu plus difficiles. Un levier active la clôture la plus proche: pour cela, on calcule la distance entre un levier et toutes les clôtures de la carte. La clôture la plus proche est désactivé lorsque le levier le plus proche est actionné. Il est aussi possible de changer ce paramètre: on peut désactiver la clôture la plus éloignée sur le même principe. Il suffit de changer :
- *maxBound :: CInt* en *minBound :: CInt* dans la fonction "actionLevier" du fichier **Model.hs**.
- L'inégalité dans la fonction auxiliaire "AuxActionLevier2" du même fichier.

- Brouillard de guerre :

Pour réaliser notre brouillard de guerre nous avons utiliser quelques astuces. Nous souhaitions obtenir un brouillard autour du personnage, de forme circulaire. Or, notre jeu est constitué de tiles (tuiles) de forme carré. Pour celà nous avons crée un spirte rectangulaire, puis dessiné un cercle dans ce rectangle.Nous avons ajouter ce sprite en intégrant notre personnage au milieu de celui-ci. Enfin nous avons ajouter un "voile" noir sur le reste de l'écran :

7 Conclusion

Ce jeu vidéo réalisé au cours du projet respecte bien certaines règles des donjon crawler. En effet nous avons bien un personnage qui peut se déplacer dans un labyrinthe, ouvrir des portes pour progresser, interagir avec certains éléments du décor et atteindre une sortie pour remporter la victoire. Nous nous sommes un peu détournée des éléments proposés dans le sujet (En terme de records) pour pouvoir proposer des extensions qui nous paraissent intéressante et ludique. Le tout respectant les conditions du sujet.

Nous avons donc réalisé un jeu vidéo en mettant au maximum en pratique les notions vues au cours du semestre avec des fonctionnalités qui sont testées avec des invariants, des pré-conditions et des post-conditions qui rendent la programmation sûre.

Nous avons fait en sorte de rendre l'ajout de fonctionnalité simple et accessible pour permettre de reprendre ce projet et d'y ajouter d'autres éléments tel qu'un système de combat avec des ennemis variés ou encore une génération de carte complexe.